IT Project Guidance

On the externalisation of Authentication and Access using Identity (IdP) and Access Management (IdAM)

**[DRAFT]**

Version:

0.2

## Purpose

The purpose of this document is to provide clear guidance to organisational leaders, IT managers, and decision-makers as to the development and operational considerations involved in externalising authentication and access using Identity Providers (IdPs) and Identity and Access Management (IdAM) systems. It seeks to illuminate the respective roles of internal and external identity solutions, while highlighting the risks associated with misplacing these responsibilities.

## Synopsis

Organisations increasingly turn to external Identity Providers (IdPs) and Identity and Access Management (IdAM) services to streamline authentication and access, aiming to reduce operational threats and leverage specialised expertise. However, the shift requires careful evaluation of how internal access controls and governance interact with external solutions. The core issue involves balancing the efficiencies of external IdAM with the necessity of maintaining robust internal oversight, ensuring that organisational security and integrity are preserved throughout the transition to externally managed identity systems.

## Contents

[Purpose 1](#_Toc207865673)

[Synopsis 1](#_Toc207865674)

[Contents 2](#_Toc207865675)

[Summary 4](#_Toc207865676)

[Introduction 6](#_Toc207865677)

[Background 7](#_Toc207865678)

[Externalising Authentication: Justifications 9](#_Toc207865679)

[Secure Bootstrapping and Transition 9](#_Toc207865680)

[Rationale for Offloading Authentication 10](#_Toc207865681)

[Security Complexity and Risk 10](#_Toc207865682)

[Operational Efficiency and Single Sign-On (SSO) 10](#_Toc207865683)

[Risks 10](#_Toc207865684)

[Clarifying Misconceptions 11](#_Toc207865685)

[Externalising User Management 11](#_Toc207865686)

[Risks 11](#_Toc207865687)

[Clarifying Misconceptions 12](#_Toc207865688)

[Externalising Access: Justifications and Boundaries 12](#_Toc207865689)

[Risks 13](#_Toc207865690)

[Clarifying Misconceptions 14](#_Toc207865691)

[Externalising Authorisation 14](#_Toc207865692)

[Why Authorisation Must Remain Internal 15](#_Toc207865693)

[Separation of Concerns 15](#_Toc207865694)

[Risks 16](#_Toc207865695)

[Clarifying Misconceptions 17](#_Toc207865696)

[Federated Identity Management 17](#_Toc207865697)

[Conclusion 18](#_Toc207865698)

[Appendices 20](#_Toc207865699)

[Appendix A - Document Information 20](#_Toc207865700)

[Versions 20](#_Toc207865701)

[Images 20](#_Toc207865702)

[Tables 20](#_Toc207865703)

[References 20](#_Toc207865704)

[Review Distribution 21](#_Toc207865705)

[Audience 21](#_Toc207865706)

[Structure 21](#_Toc207865707)

[Diagrams 21](#_Toc207865708)

[Acronyms 21](#_Toc207865709)

[Terms 22](#_Toc207865710)

[Appendix B – Literature Evidence 22](#_Toc207865711)

[Guidance Supporting Internal User Management 22](#_Toc207865712)

[Microsoft Entra ID: Authentication vs. Application Logic 22](#_Toc207865713)

[Microsoft Entra ID: Application Access Management 23](#_Toc207865714)

[Microservices Best Practice: Local User Context 23](#_Toc207865715)

[Federation Patterns: Linking External Identity to Internal User 23](#_Toc207865716)

[Security Risks in Distributed Systems 24](#_Toc207865717)

[Appendix C - Microservices and Authorisation Autonomy 25](#_Toc207865718)

[Appendix D: Microservices, Duplication, and Architectural Trade-offs 26](#_Toc207865719)

# Summary

In the broader era of cloud service provider-driven development, there is a pervasive tendency to misidentify the boundaries of system capabilities—often leading to the inappropriate classification as to whether they are internal or external. This confusion can introduce significant operational fragility, increase the burden of ongoing effort, and drive up long-term costs. The impact is particularly acute in the realm of identity services, where misplaced boundaries jeopardise both system autonomy and security. For these reasons, it is essential to clearly distinguish which identity capabilities should be externalised, which must remain internal, and to understand why these lines of responsibility are fundamental to robust, maintainable architecture.

The document identifies and discusses four of these capabilities —session, authentication, access, and authorisation—supported by three others -- user management, role management, and permission management. The distinctions between each are critical to making sound architectural decisions and avoiding the conflation of fundamentally different concerns.

Each capability is examined in turn:

* **Authentication** is high-risk and operationally complex. Offloading it to a trusted Identity Provider (IdP) is best practice, provided the system retains internal user records and auditability. Authentication verifies identity, but does not govern user behaviour or system access.
* **User Management** must remain internal. Systems must maintain their own user records to support domain-specific logic, enforce access control, and provide auditability. Tokens may assert identity, but they do not reflect dynamic state, delegated authority, or operational context.
* **Access Control** can be externalised, but only at the perimeter and only for coarse-grained decisions. Infrastructure components such as WAFs, API gateways, and IdPs are well-suited to enforcing organisational policies. However, attempts to model system-specific roles externally introduce overreach and entanglement.
* **Authorisation** must remain internal. It governs what a user is allowed to do within the system and is tightly coupled to business logic, data structures, and operational state. Externalising authorisation leads to semantic drift, coordination overhead, and loss of fidelity.

Throughout the document, vendor-driven architecture patterns are critically examined. Cloud platforms often promote patterns that reflect their capabilities and commercial positioning, with less concern to alignment to enduring system design principles. While platform features may enable externalisation, they do not define architectural responsibility. Best practice must be grounded in system needs, not in the roadmap or integration convenience of the hosting platform.

It is worth noting, however, that vendor documentation—particularly from providers such as Microsoft—does include best practice guidance that aligns with sound architectural principles. These documents often caution against overreach, recommend separation of concerns, and acknowledge the limits of external identity systems. Relevant examples are captured in Appendix B.

The paper also highlights the importance of maintaining architectural clarity. Identity services may govern who can access a system, but only the system itself can determine what that user is allowed to do. This separation of concerns is not merely a design preference—it is a safeguard against operational entanglement, audit failure, and security drift.

Maintaining these boundaries ensures that systems remain secure, auditable, and adaptable—particularly in regulated environments where accountability and traceability are non-negotiable. It also supports maintainability, reduces coordination overhead, and preserves system autonomy in the face of evolving business requirements and platform changes.

This paper underscores the necessity for architectural decisions to be grounded in robust, enduring design principles, even amid the adoption of advanced technologies such as Federated Identity (FID). While FID modern identity platform capabiities offer significant capabilities to externalise certain aspects of authentication and access control, they do not diminish the system architect’s responsibility to maintain critical boundaries. The paper demonstrates—through patterns like FID—that the presence of these externalised technologies must not override the foundational requirement for system autonomy, auditability, and integrity. Instead, FID should be leveraged within its intended scope: to facilitate secure *authentication* and perimeter-level *access* control, while ensuring that user management and *authorisation* remain internal and deeply integrated with business logic. Only by retaining this separation of concerns can organisations avoid operational entanglement and ensure that new identity platforms serve the system’s integrity, rather than replace it.

Finally, the paper calls for a shift in architectural consideration: away from feature-led integration and toward responsibility-led design. Identity platforms are valuable tools, but they must be used in service of system integrity—not as substitutes for it.

# Introduction

This paper addresses a recurring architectural challenge in modern system design: the misapplication of identity and access management patterns, particularly in relation to authentication, access control, user management, and authorisation.

As organisations increasingly adopt cloud-native platforms and integrate with enterprise identity providers, there is a growing tendency to externalise core system capabilities. While some externalisation is appropriate—indeed, necessary—other forms introduce architectural misalignment, operational fragility, and long-term cost.

The problem is not the availability of external identity services, but the assumption that their presence implies architectural responsibility. Vendor platforms offer extensive tooling to support authentication, access control, and role management. However, these tools are often promoted without sufficient regard for system boundaries, domain-specific logic, or auditability requirements.

This paper sets out to clarify:

* Which identity-related capabilities can be safely externalised,
* Which must remain internal to the system,
* And why the distinction matters for maintainability, security, and accountability.

It is written for architects, technical leads, and decision-makers working in environments where system autonomy, auditability, and operational resilience are non-negotiable—particularly in regulated sectors such as government, finance, and healthcare.

The analysis is grounded in practical patterns, vendor tooling, and platform behaviours, but is framed by architectural principles. It aims to support informed decision-making, reduce misalignment between platform capabilities and system responsibilities, and promote enduring, maintainable system design.

**Federated Identity Management (FIM)**

While this paper focuses primarily on emerging patterns that externalise or offload authorisation logic, it also touches on established architectural approaches where their use has drifted beyond original intent. One such example is Federated Identity Management (FIM). FIM is a mature and widely adopted pattern for cross-boundary authentication, particularly in public sector and education contexts. However, when misunderstood or overextended into the domain of authorisation—effectively becoming a form of Federated Authorisation Management—it can introduce the same risks this paper seeks to highlight. The implications of this will be explored in a later section, after covering foundational concepts that provide appropriate scaffolding for evaluating FIM and other identity and access design patterns.

# Background

Modern systems—whether monolithic, modular, or microservice-based—require a consistent set of foundational capabilities to function securely and reliably. These include but are not limited to configuration management, diagnostics, routing, storage, caching, session management, user management, permissions management, roles management, authentication of session users, authorisation of system users’ access to system managed resources.

A close-up of a screen

AI-generated content may be incorrect.

Figure 1: Example core system capabilities

Many of these capabilities are considered core and must be consciously sourced as part of the system design before tackling client business specific capabilities.

They can be:

* *bought* as part of a system, platform or framework,
* *built* in-house as part of a custom system,
* *enhanced* by integration with external, specialised, services,
* *offloaded* entirely to an external service, or
* *undelivered*, the latter constituting a failure state.

Among these, **session management**, **authentication**, **access**, and **authorisation** are frequently both incorrectly understood conflated as well as misunderstood in terms of what can be safely offloaded or omitted, These represent distinct layers of responsibility:

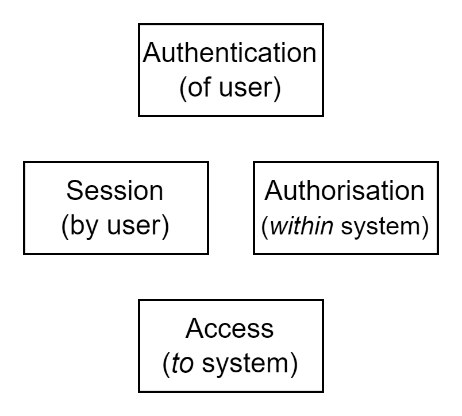


Figure 2: Session, Access, Authentication, Authorisation

* **Session** refers to the system’s mechanism for maintaining a user’s active presence. It allows the system to associate incoming requests with a specific user context, either through stateful tracking or stateless tokens.
* **Authentication** is the process of verifying the identity of an external actor and associating it with a session. It answers the question: *Who is this?*
* **Access** determines whether a user is permitted to enter the system at all. This is typically enforced at the perimeter—via an Identity Provider (IdP) or gateway—and may include conditional access policies such as device compliance or location.
* **Authorisation** governs what a user is allowed to do within the system. It is fine-grained, domain-specific, and tightly coupled to internal logic and data.

Understanding these distinctions is critical to making sound architectural decisions. While authentication and access can often be offloaded to an IdP, session management and authorisation must remain within the system to ensure operational integrity, auditability, and domain-specific control.

These four runtime concepts in turn rely on a set of supporting capabilities that must also be present in the system:

* **User Management** maintains internal records of users, including identifiers, status, and domain-specific attributes. It supports authentication (by linking external identities to internal users), access (by enabling or disabling users), and authorisation (by assigning roles and permissions).
* **Permission Management** defines the specific actions permitted on resources. It is the most granular layer of authorisation and must be tightly coupled to system logic.
* **Role Management** defines groupings of permissions that can be assigned to users. It simplifies the assignment and interpretation of access rights.

*Authentication* is particularly high-risk due to the complexity of securely managing credentials and the evolving standards around identity verification. As a result, it is now widely accepted best practice to offload authentication to a trusted third-party IdP, such as Microsoft Entra ID (formerly Azure AD), Okta, or Auth0. These services provide secure, standards-based identity verification using protocols such as OpenID Connect (OIDC) and SAML 2.0.

However, even when *authentication* is offloaded, *user management* is not. Systems must maintain internal user records to support domain-specific logic, auditing, and access control. The IdP provides a trusted assertion of identity, but the system must determine which internal user record to associate with that identity. This internal user record is the basis for auditability and enforcement of system-specific policies.

It is important to distinguish between vendor-promoted patterns and architecture aligned to system responsibility. Cloud service providers often promote patterns that reflect the capabilities and commercial positioning of their platforms, rather than enduring system design principles. In particular, patterns that encourage deep reliance on external identity platforms for internal access control introduce operational entanglement, reduce autonomy, and increase long-term cost. These patterns may offer integration convenience or alignment with platform-native tooling, but they do so at the expense of maintainability, auditability, and domain-specific control. Best practice must be defined by system needs—not by the feature set or roadmap of the hosting platform.

User management within the system is also the foundation for authorisation. While authorisation can be enhanced by external services (e.g., by receiving coarse-grained roles from an IdAM), it must not be offloaded. Authorisation is inherently domain-specific and must be implemented within the system using modifiable data structures.

Microsoft’s own guidance reinforces this distinction: while Entra ID can provide IdP services to support authentication and IdAM services to provide access control and coarse-grained role claims, applications are expected to manage, interpret, and enforce access control internally.

# Externalising Authentication: Justifications

While many system capabilities must remain internal, **authentication** is a notable exception where offloading is not only common but widely considered best practice. This is due to a combination of historical failures, operational risk, and the increasing complexity of secure identity verification.

## Secure Bootstrapping and Transition

In-system authentication—where the system stores and verifies user credentials—remains a valid pattern during initial deployment. It is often used to allow operational administrators to install and configure the system, including setting up integration with an external Identity Provider (IdP). Once this integration is established and tested, internal authentication is typically disabled. This staged approach of first extension then sole reliance is well-established and supports secure bootstrapping.

## Rationale for Offloading Authentication

The rationale for first extending then relying solely on external authentication is twofold:

### Security Complexity and Risk

There is a long and well-documented history of systems failing to secure credential storage. This includes poor hashing practices, lack of salting, insecure password resets, and failure to adopt evolving standards such as multi-factor authentication (MFA), device trust, and zero trust principles. The cost of maintaining secure authentication logic is high, compounded by having to repeat it in multiple systems, and the risk of compromise is significant. A mature market exists both to exploit these weaknesses and to sell remediation services.

### Operational Efficiency and Single Sign-On (SSO)

Organisations increasingly rely on IdPs such as Microsoft Entra ID, Okta, or Auth0 to provide single sign-on across multiple systems. This reduces the burden on users and administrators, improves compliance, and enables centralised identity lifecycle management. Offloading authentication to a trusted IdP allows systems to benefit from continuous security updates, threat intelligence, and compliance certifications that would be infeasible to maintain to the same degree in-house, across all systems.

## Risks

While offloading authentication is best practice, it must be done with clear boundaries. The following risks emerge when authentication is treated as a substitute for other identity-related capabilities:

**Loss of Internal Anchoring**  
If authentication is offloaded without retaining internal user records, the system loses the ability to associate identity with domain-specific logic, permissions, and audit trails.

**Overreliance on Token Semantics**  
Tokens issued by IdPs may carry identity claims, but they do not reflect dynamic state, delegated authority, or system-specific roles. Relying solely on token contents for access decisions introduces fragility and misalignment.

**Auditability Gaps**  
Authentication events handled externally may not be recorded in a way that supports system-level audit requirements. Without internal linkage to user records and session context, traceability is compromised.

## Clarifying Misconceptions

It is a common misconception that authentication and user management are interchangeable. Authentication verifies identity; user management governs the lifecycle, attributes, and operational state of users within the system. Offloading the former does not eliminate the need for the latter.

Another misconception is that identity tokens can serve as a complete substitute for internal access control. While tokens are useful for conveying identity and coarse-grained roles, they do not support fine-grained permissions, dynamic state, or provide a stable audit anchor.

As will be discussed in the next section, **user management must remain internal**. It is foundational to authorisation, auditability, and operational control. Systems must maintain their own user records, even when authentication is externally handled.

# Externalising User Management

User management is a core system capability that must remain internal. It is often conflated with authentication, but the two serve different purposes. Authentication verifies identity; user management governs the lifecycle, attributes, and operational state of users within the system.

While authentication can be offloaded to an external Identity Provider (IdP), user management cannot. The system must maintain its own user records to support domain-specific logic, enforce access control, and provide auditability. A system user can be disabled independently of the user’s status in the IdP. Each session and operation performed within the system must be recorded against an internal user identity to support accountability.

## Risks

Despite the operational appeal of centralising identity, offloading user management introduces significant risks that compromise system autonomy, traceability, and resilience.

**Loss of System-Level Control**  
Without internal user records, the system cannot independently disable or restrict users. Access becomes entirely dependent on the IdP’s state, which may not reflect system-specific concerns.

**Fragile Audit Trails**  
Audit records tied only to external identities are brittle. Changes in the IdP—such as renaming or deletion—can break continuity and undermine accountability.

**Lack of Domain-Specific Attributes**  
Systems often require user attributes not present in the IdP (e.g., onboarding status, delegated authority). Without internal records, these must be managed externally, increasing complexity and reducing clarity.

**Reduced Resilience**  
IdP outages or misconfigurations can render the system inaccessible. Migration between IdPs becomes harder without internal user anchors to maintain continuity and traceability.

**Over-Coupling to External Structures**  
Aligning internal logic to IdP roles or groups introduces tight coupling. Changes in external naming or structure can break internal logic and require coordinated updates across systems

## Clarifying Misconceptions

It is a common error to treat authentication and user management as interchangeable. Authentication verifies identity; user management tracks and governs system users. Tokens may assert identity, but they do not replace the need for internal user records, nor do they support dynamic state, delegated authority, or auditability.

Another misconception is that diagnostic tracing can substitute for user-level auditability. While cloud-native platforms offer powerful tools for aggregating logs and tracing activity, these are designed for operational diagnostics—not for compliance-grade audit trails. Auditability requires durable, tamper-evident records tied to internal user identities and retained for mandated durations.

The notion that internal user management is outdated reflects a vendor-centric view. From a platform perspective, externalising identity simplifies integration. But from an enterprise perspective—particularly in regulated environments such as government agencies—internal user records are essential. They enable enforcement of domain-specific policies, support audit, and maintain operational independence.

# Externalising Access: Justifications and Boundaries

*Access* control—deciding whether a user may reach a system or specific parts of it—is a capability that is routinely externalised across modern architectures. It is often enforced by multiple layers of infrastructure, including:

* **Web Application Firewalls (WAFs)**, which restrict access based on IP ranges, geolocation, or request patterns,
* **API gateways**, which enforce authentication and rate limiting before requests reach backend services,
* **Identity Providers (IdPs)**, which issue tokens only to users who meet organisational access policies.

This layered approach is logical and well-established. It allows access control to be applied at the perimeter, before any business logic is invoked. It also supports enterprise-wide policies such as:

* Differentiating employees from contractors or external partners,
* Restricting access to systems based on business unit or team membership,
* Enforcing conditional access policies (e.g., device compliance, location, time-of-day).

These checks are *coarse*-grained and *organisational* in nature – not *system* in nature. They are best handled by external systems that specialise in perimeter enforcement and identity verification. This is where offloading makes sense: **access control at the infrastructure and enterprise boundary**.

However, the scope of access control ***must*** remain appropriately coarse. It is suitable for distinctions such as “Employee” vs “Contractor”, or “Finance Department” vs “External Vendor”. It is not suitable for system-specific roles such as “Architect”, “Contract Writer”, or “Invoice Approver”. These roles are not organisational—they are contextual, operational, and tied to the logic of the system itself.

Attempting to encode such roles in external access control systems introduces overreach. It requires external systems to understand and model internal business logic, which they are not designed to do. This leads to brittle dependencies, increased coordination overhead, and long-term operational cost.

## Risks

While externalising access is appropriate at the perimeter, extending it too far introduces architectural and operational risks:

**Loss of Boundary Clarity**: When access control systems begin to encode business or system roles, the boundary between infrastructure and application logic becomes blurred. This undermines separation of concerns and increases the risk of misconfiguration.

**Over-Engineering and Role Inflation**: External systems are not designed to model the internal semantics of a service. Attempting to do so leads to role inflation—where external roles proliferate to reflect internal logic—and over-engineering, where the IdAM or gateway becomes a pseudo-application.

**Operational Entanglement**: Encoding system-specific access logic in external systems introduces dependencies between teams, systems, and release cycles. Changes to business logic require coordinated updates across infrastructure and identity platforms, increasing overhead and reducing agility.

**Loss of Flexibility**: IdAM systems are designed to model organisational identity and access—not system-specific behaviour. Attempting to use them to represent dynamic constructs such as temporary role assignments, delegated authority, or resource-specific access introduces rigidity. These constructs evolve frequently and require in-system logic to remain responsive and maintainable.

**Loss of Fidelity:** Even when external systems can technically represent internal roles or permissions, they often do so imprecisely. The semantics of a system role—its scope, conditions, and constraints—are difficult to express in external identity structures. This leads to misalignment, over-permissioning, and reduced confidence in access control correctness.

**Reduced Auditability and Traceability**: Access decisions made outside the system may not be recorded in a way that supports internal audit requirements. This creates gaps in traceability and undermines accountability.

**Security Drift**: External roles reused across systems may grant broader access than intended. Without tight coupling to system logic, the principle of least privilege is easily violated.

## Clarifying Misconceptions

It should now be clear that **Access is not Authentication**. Access refers to the decision of whether a user may reach a system or its components, typically enforced at the perimeter. Authentication, by contrast, is the process of verifying the identity of the user once access is attempted.

This distinction is critical. Conflating access with authentication leads to architectural missteps—such as assuming that a token issued by an IdP is sufficient to determine what a user can do within the system. It is not.

Architects must recognise that:

* **Access control** can be externalised, but must remain coarse and infrastructure-scoped.
* **Authentication** can be offloaded, but must be linked to internal user records.
* **Authorisation** must remain internal, as it is tightly coupled to system logic and operational context.

Maintaining these boundaries ensures that systems remain auditable, adaptable, and aligned to their domain-specific responsibilities. It also avoids the operational entanglement and security drift that result from overreliance on external identity platforms.

# Externalising Authorisation

As previously covered in the discussion on externalising access control, perimeter-level decisions—such as whether a user may reach a system or its components—can be safely offloaded to infrastructure and identity platforms. These decisions are coarse-grained, organisational in nature, and well-suited to enforcement by Web Application Firewalls (WAFs), API gateways, and Identity Providers (IdPs).

**Authorisation**, however, is a distinct capability. It governs what a user is allowed to do *within* the system. It is fine-grained, domain-specific, and tightly coupled to business logic, data structures, and operational context. It is typically implemented by associating users with roles, and roles with permissions, which are then enforced at runtime to control access to resources, actions, or data.

While some IdAM platforms support role definitions, group memberships, and token-based delivery of claims, these mechanisms are suitable only for coarse-grained access decisions. Attempting to externalise authorisation—by embedding system-specific roles or entitlements into identity tokens—conflates enterprise-wide access with system-specific control. This introduces architectural and operational risks that cannot be mitigated by platform features alone.

## Why Authorisation Must Remain Internal

Once a user is authenticated and granted access to the system, the question becomes: *what are they allowed to do within this system?* This is where authorisation becomes domain-specific and must be implemented internally.

System roles and permissions are not static. They evolve due to:

* Organisational restructures,
* Temporary assignments (e.g., acting roles),
* Delegated authority,
* Changes in business logic or policy.

Hardcoding these roles or relying on external tokens to define them introduces rigidity. Instead, systems must provide interfaces to assign and manage roles dynamically—ideally with support for time-bound assignments, delegation, and auditability. This is not trivial, but it is well-supported by modern development frameworks (e.g., .NET Core’s policy-based authorisation model).

External IdAMs do not support this level of granularity. They do not understand the internal structure of the system, the semantics of its roles, or the context in which permissions are granted. Attempting to encode this logic externally creates brittle dependencies, operational distance, and increased risk.

## Separation of Concerns

The correct design pattern is a **separation of concerns**:

* The IdAM governs *who* can access the system, and under what enterprise-wide conditions.
* The system governs *what* that user can do once inside, based on its own domain model and internal user state.

Blurring this boundary—by tying system-level authorisation to IdAM roles—creates a poor architectural pattern. It places operational responsibility for system access logic into a different system, often managed by a different team, with different priorities and release cycles, and most crucially lack of knowledge of the business system’s purpose, users, processes. While it can be encoded in standard procedure documentation, it almost immediately is out of date – or worse, removes flexibility from the business group. This undermines agility, increases coordination costs, and introduces security risks when roles are reused or misinterpreted.

## Risks

While superficially attractive, offloading authorisation introduces a number of systemic risks:

**Loss of Domain Semantics**: Authorisation decisions depend on the semantics of the system: what constitutes a resource, what actions are permitted, and under what conditions. External IdAM systems cannot model this complexity effectively. Attempting to encode domain-specific permissions in a centralised system leads to either:

* Over-permissioning, where roles are too broad to be safe, or
* Over-engineering, where the IdAM becomes a brittle, pseudo-application.

**Coordination Overhead and Semantic Drift**: Externalising authorisation requires tight coordination between service teams and IdAM administrators. Role definitions must be synchronised across systems, and any change to role semantics must be reflected in both the IdAM and the consuming service. This introduces versioning complexity and increases the risk of semantic drift.

**Token Bloat and Fragility**: Embedding detailed authorisation data in tokens increases their size and complexity. This can lead to infrastructure issues (e.g., header size limits), security risks (e.g., token leakage), and operational fragility (e.g., difficulty revoking or updating permissions without reissuing tokens).

**Inflexibility in Evolving Access Models**: As systems evolve, so do their access control requirements. Externalised authorisation logic is difficult to adapt to new models such as resource-based access, contextual policies, or delegated authority. These require internal logic and dynamic evaluation.

**Undermining of Least Privilege**: External roles tend to be coarse-grained and reused across services. This undermines the principle of least privilege, as users are often granted access to more functionality than required, simply because the role is shared or overloaded.

**Reduced Auditability and Traceability**: Authorisation decisions made outside the system may not be recorded in a way that supports internal audit requirements. This creates gaps in traceability and undermines accountability.

## Clarifying Misconceptions

It is often assumed that because authentication and access control can be externalised, authorisation can follow the same pattern. This is a misconception. Authorisation is not a continuation of authentication—it is a distinct capability with different requirements, responsibilities, and risks.

External identity platforms are designed to verify identity and enforce organisational access policies. They are not designed to model system-specific behaviour, interpret business logic, or enforce fine-grained permissions. Treating external roles or token claims as sufficient for authorisation leads to architectural misalignment and operational fragility.

Another common misunderstanding is that embedding roles or entitlements in identity tokens provides a complete and portable access model. In reality, tokens are snapshots of identity context at a point in time. They do not reflect dynamic state, cannot adapt to evolving business logic, and are difficult to revoke or update without reissuing. They are useful for bootstrapping access, but not for enforcing system-level authorisation.

Architects must recognise that:

* **Authentication** verifies identity and can be offloaded.
* **Access control** governs entry and can be externalised, but must remain coarse.
* **Authorisation** governs capability and must remain internal, as it is tightly coupled to system logic, user state, and operational context.

Maintaining these boundaries ensures that systems remain secure, auditable, and adaptable. It also avoids the entanglement, drift, and overreach that result from treating external identity platforms as substitutes for internal access control logic.

# Federated Identity Management

Federated Identity Management (FIM) is a mature and widely adopted pattern that enables users to authenticate across organisational boundaries using a single digital identity. It is particularly common in public sector and education contexts, where interoperability and user convenience are operational priorities.

FIM typically relies on trust frameworks and standardised protocols—such as SAML, OAuth2, or OpenID Connect—to allow identity providers (IdPs) to assert claims about users to relying systems.

Within its intended scope—authentication of identity—FIM is effective. It reduces credential sprawl, centralises identity assurance, and supports scalable access to distributed systems.

However, its increasing use as a mechanism for *authorisation* introduces architectural risks that are often overlooked. These risks are not inherent to FIM itself, but arise when identity claims are treated first for *access,* then slipping into authorisation control decisions within consuming systems.

This misapplication—effectively a form of Federated *Authorisation* Management rather than Federated *Identity* Management—blurs the boundary between identity verification and permission enforcement. It assumes that externally assigned roles or group memberships can reliably substitute for system-specific authorisation logic. In practice, this leads to coarse-grained access control, brittle mappings, and a loss of contextual nuance. It also introduces dependencies on external role definitions and trust relationships, which may not reflect the operational needs or domain semantics of the system.

The risks associated with this pattern mirror those outlined in the previous section on offloading authorisation. The difference is that FIM, due to its maturity and ubiquity, is often adopted without scrutiny. This paper does not argue against the use of FIM for authentication, or very coarse access control. Rather, it cautions against the overextension of *access* into *authorisation*, and the architectural consequences that follow.

Finally, while true federated identity is appropriate in *multi*-organisational contexts – hence *federation* -- it is sometimes adopted—or invoked—within *single*-organisation systems where it is not required. In such cases, the term “federation” may be used to describe what is actually centralised identity and access management. This can obscure architectural boundaries and lead to overreach, particularly when coarse-grained roles are treated as sufficient for system-level authorisation – but also a higher delay and cost of departmental coordination and operations for the whole service lifespan.

# Conclusion

Identity and access capabilities—specifically authentication, access control, and authorisation—are foundational to secure system design. While modern platforms offer extensive tooling to externalise these capabilities, doing so without regard for architectural boundaries introduces risk, rigidity, and operational overhead.

This paper has clarified the distinctions between session, authentication, access, and authorisation, and has examined each in turn:

* **Authentication** is high-risk and operationally complex. Offloading it to a trusted Identity Provider is best practice, provided internal user records and auditability remain in place.
* **Access control** can be externalised, but only at the perimeter and only for coarse-grained decisions. Attempts to model system-specific roles externally lead to overreach and entanglement.
* **Authorisation** must remain internal. It is tightly coupled to system logic, user state, and business context. Externalising it undermines agility, fidelity, and accountability.

Architects must resist the temptation to treat vendor capabilities as architectural guidance. Platform features may enable externalisation, but that does not make it appropriate. Best practice is defined by system responsibility, not by integration convenience.

Maintaining clear boundaries between organisational identity services and system-specific access logic ensures that systems remain secure, auditable, and adaptable. It also preserves autonomy, reduces coordination overhead, and supports long-term maintainability.

Appendices

Appendix A - Document Information

Authors & Collaborators

* Sky Sigal, Solution Architect

### Versions

* 1. Initial Draft
  2. Updated to refer to FID patterns.

### Images

[Figure 1: TODO Image 2](#_Toc144995112)

### Tables

[Table 1: TODO Table 3](#_Toc145048484)

[Table 2: TODO Table 2 3](#_Toc145048485)

### References

**There are no sources in the current document.**

1. Microsoft Learn – [Authenticate applications and users with Microsoft Entra ID](https://learn.microsoft.com/en-us/entra/architecture/authenticate-applications-and-users)
2. Microsoft Learn – [Azure identity & access security best practices](https://learn.microsoft.com/en-us/azure/security/fundamentals/identity-management-best-practices)
3. Microsoft Learn – [Best practices for securely deploying Microsoft Entra ID Governance](https://learn.microsoft.com/en-us/entra/id-governance/best-practices-secure-id-governance)
4. Nblocks – [User Management in Microservices: 10 Best Practices](https://www.nblocks.dev/blog/user-management/user-management-microservices)
5. CISA – [Identity and Access Management Recommended Best Practices](https://www.cisa.gov/sites/default/files/2023-12/ESF%20IDENTITY%20AND%20ACCESS%20MANAGEMENT%20RECOMMENDED%20BEST%20PRACTICES%20FOR%20ADMINISTRATORS%20PP-23-0248_508C.pdf)
6. CodezUp – [Authentication without Sessions: JWT and Sessionless Web Applications](https://codezup.com/auth-without-sessions-jwt-and-sessionless-web-applications/)

### Review Distribution

The document was distributed for review as below:

|  |  |
| --- | --- |
| Identity | Notes |
| Duncan Watson, Enterprise Architect |  |
| Mike O’Connor Enterprise Security Architect |  |
| Amy Orr, Enterprise Data Architect |  |
| Gareth Philpott, Solution Architect |  |
| Carl Klitscher, Solution Architect |  |

### Audience

The document is technical in nature, but parts are expected to be read and/or validated by a non-technical audience.

### Structure

Where possible, the document structure is guided by either ISO-\* standards or best practice.

### Diagrams

Diagrams are developed for a wide audience. Unless specifically for a technical audience, where the use of industry standard diagram types (ArchiMate, UML, C4), is appropriate, diagrams are developed as simple “box & line” monochrome diagrams.

### Acronyms

API

: [Application Programming Interface](#Term_ApplicationProgrammingInterface).

DDD

: Domain Driven Design

GUI

: [Graphical User Interface](#Term_ApplicationProgrammingInterface). A form of [UI](#Acronym_UI).

ICT

: acronym for Information & Communication Technology, the domain of defining Information elements and using technology to automate their communication between entities. [IT](#Acronym_IT) is a subset of ICT.

IT

: acronym for Information, using Technology to automate and facilitate its management.

UI

: User Interface. Contrast with [API](#Acronym_API).

### Terms

Refer to the project’s Glossary.

Application Programming Interface

: an Interface provided for other systems to invoke (as opposed to User Interfaces).

Capability

: a capability is what an organisation or system must be able to achieve to meet its goals. Each capability belongs to a domain and is realised through one or more functions that, together, deliver the intended outcome within that area of concern.

Domain

: a domain is a defined area of knowledge, responsibility, or activity within an organisation or system. It groups related capabilities, entities, and functions that collectively serve a common purpose. Each capability belongs to a domain, and each function operates within one.

Entity

: an entity is a core object of interest within a domain, usually representing a person, place, thing, or event that holds information and can change over time, such as a Student, School, or Enrolment.

Function

: a function is a specific task or operation performed by a system, process, or person. Functions work together to enable a capability to be carried out. Each function operates within a domain and supports the delivery of one or more capabilities.

Person

: a physical person, who has one or more Personas. Not necessarily a system User.

Persona

: a facet that a Person presents to a Group of some kind.

Quality

: a quality is a measurable or observable attribute of a system or outcome that indicates how well it meets expectations. Examples include reliability, usability, and performance. Refer to the ISO-25000 SQuaRE series of standards.

User

: a human user of a system via its UIs.

User Interface

: a system interface intended for use by system users. Most computer system UIs are Graphics User Interfaces ([GUI](#Acronym_GUI)) or Text/Console User Interfaces (TUI).

Appendix B – Literature Evidence

### Guidance Supporting Internal User Management

#### Microsoft Entra ID: Authentication vs. Application Logic

Microsoft Entra ID (formerly Azure AD) provides authentication services and coarse-grained access control (e.g., app roles, group membership). However, Microsoft explicitly states that **applications are responsible for interpreting and enforcing access control** based on the identity information received.

“After apps receive the token, they can use the information in that token to identify the user.”  
“Applications interact with Microsoft Entra ID tenants to authenticate users... Microsoft Entra ID redirects the user back to the application, with or without a token.”  
— [Microsoft Learn: Authenticate applications and users with Microsoft Entra ID](https://learn.microsoft.com/en-us/entra/architecture/authenticate-applications-and-users)

1

This implies that the application must maintain a user record to associate the token with internal logic, permissions, and auditability.

#### Microsoft Entra ID: Application Access Management

Microsoft Entra ID supports assigning users and groups to applications, but this is **not a substitute for internal user management**. It is a mechanism for controlling *who can access* an application—not *what they can do once inside*.

“With certain types of applications, you have the option of requiring users to be assigned to the application. By doing so, you prevent everyone from signing in except those users you explicitly assign.”  
— [Microsoft Learn: Manage access to apps](https://learn.microsoft.com/en-us/entra/identity/enterprise-apps/what-is-access-management)

2

This reinforces the idea that Entra ID governs *entry*, but the application must govern *behaviour*.

#### Microservices Best Practice: Local User Context

In microservice architectures, best practice is for each service to maintain its own user context, including roles and permissions. This is necessary for domain-specific logic, auditability, and resilience.

“Each microservice operates independently, making it difficult to maintain a consistent and secure approach to managing users... Implement strict access controls on all API endpoints... Design communication protocols for permission data.”  
— [Nblocks: User Management in Microservices](https://www.nblocks.dev/blog/user-management/user-management-microservices" \t "_blank)

3

#### Federation Patterns: Linking External Identity to Internal User

Federated identity patterns (e.g., using OIDC or SAML) explicitly recommend **linking external identities to internal user records**. This is necessary for session continuity, audit trails, and consistent user experience.

“IAM receives the Access Token and ID Token, and uses the user identifier (sub) to link the external user to the internal user... The IAM system may keep mapping the user claims from the external IdP to the internal user account during each login.”  
— [CerberAuth: Federating an Identity Provider](https://www.cerberauth.com/blog/identity-federation/" \t "_blank)

4

#### Security Risks in Distributed Systems

Failing to maintain internal user records increases the risk of unauthorised access, data leakage, and audit failure. Distributed systems are particularly vulnerable when identity and access control are not tightly scoped and enforced.

“Weak authentication and authorisation: If these are weak, unauthorized people might get access to sensitive areas of the system... Without proper security, unauthorized users could gain access to the system, leading to data theft.”  
— [GeeksforGeeks: Vulnerabilities in Distributed Systems](https://www.geeksforgeeks.org/computer-networks/vulnerabilities-and-threats-in-distributed-systems/" \t "_blank)

**Federated Identity Management Framework (FIMF) for Civilian Agencies – ATARC**

This framework by the Advanced Technology Academic Research Center (ATARC) emphasizes the importance of a shared responsibility model for identity federation. It aligns with the document's caution against overextending federated identity into authorisation

[fimf-for-civilian-agencies.pdf](https://atarc.org/wp-content/uploads/2025/04/fimf-for-civilian-agencies.pdf)

[atarc.org/wp-content/uploads/2025/04/fimf-for-civilian-agencies.pdf](https://atarc.org/wp-content/uploads/2025/04/fimf-for-civilian-agencies.pdf)

[aircconline.com/csit/papers/vol11/csit110502.pdf](https://aircconline.com/csit/papers/vol11/csit110502.pdf)

[FID\_WP\_0504](https://educationgovtnz.sharepoint.com/sites/FSISGroup/Shared%20Documents/Forms/AllItems.aspx?id=%2Fsites%2FFSISGroup%2FShared%20Documents%2FX%20ESAF%2F02%20ESFS%20%2D%20Federated%20Search%2F01%20ESAA%20%2D%20Auth%20and%20Auth%2FNew%20file%20system%20ESAA%20STAGE%202%2FArchive%2FFederated%20Identity%2FFID%5FWP%5F0504%2Epdf&parent=%2Fsites%2FFSISGroup%2FShared%20Documents%2FX%20ESAF%2F02%20ESFS%20%2D%20Federated%20Search%2F01%20ESAA%20%2D%20Auth%20and%20Auth%2FNew%20file%20system%20ESAA%20STAGE%202%2FArchive%2FFederated%20Identity)   
This white paper by RSA Security discusses the benefits of federated identity, including strong authentication and single sign-on (SSO) functionality. It supports the document's assertion that federated identity can be effective for authentication but should not be overextended into authorisation 

1. Federated Identity Management Framework (FIMF) for Civilian Agencies - ATARC 5
2. Federated Identity Management (FIdM) Systems Limitation and Solutions - Maha Aldosary and Norah Alqahtani 6
3. Flexible federated identity credentials (preview) - Microsoft Entra 7
4. ERS Solutions Architecture v1.8 - [Vikas.Awasthi@nz.unisys.com](mailto:Vikas.Awasthi@nz.unisys.com); David Barber 2
5. ERS Solutions Architecture v2.6  
   [Vikas.Awasthi@nz.unisys.com](mailto:Vikas.Awasthi@nz.unisys.com);   
   David Barber 3
6. FID\_WP\_0504 - Brian Breton; S-1-5-21-1645522239-261903793-725345543-33285 4

Appendix C - Microservices and Authorisation Autonomy

A common critique of decentralised authorisation is that it leads to divergence in logic across microservices. This is often framed as a risk—suggesting inconsistency, policy drift, or increased attack surface. However, this critique misunderstands the purpose and design philosophy of microservices.

Microservices are intentionally autonomous. Each service is designed to encapsulate its own domain logic, evolve independently, and be deployed without coordination delays. Divergence is not a flaw—it is a feature. It reflects the reality that different services have different responsibilities, different data models, and different access control requirements.

Attempting to centralise authorisation—whether through a shared microservice or via an external IdAM—introduces the very entanglement that microservices are designed to avoid. It requires all services to align on a common role model, a shared permission vocabulary, and a coordinated release process. This undermines agility, increases coupling, and creates a bottleneck for change.

Moreover, centralised authorisation logic often becomes a pseudo-application in its own right—requiring its own governance, testing, and deployment lifecycle. It must be updated whenever any service changes its access model, leading to cascading dependencies and increased operational overhead.

The same critique applies to the use of external IdAM systems to enforce fine-grained authorisation. While IdAM platforms are well-suited to coarse-grained access control (e.g., “Finance team” vs “External vendor”), they are not designed to model system-specific roles such as “Contract Approver” or “Invoice Reviewer.” Encoding such roles externally requires the IdAM to understand internal business logic, which introduces fragility and misalignment.

Microservices should define and enforce their own authorisation logic, using internal role and permission models that reflect their domain. They may share identity context (e.g., user ID, organisation ID), but they must retain control over how that context is interpreted and enforced.

This approach supports:

* **Autonomy**: Services evolve independently without coordination bottlenecks.
* **Fidelity**: Authorisation logic reflects the semantics of the service, not a generic external model.
* **Resilience**: Changes to one service do not require changes to others or to a centralised authorisation layer.
* **Auditability**: Each service can record and trace its own access decisions, aligned to its own logic.

In short, divergence in authorisation logic across microservices is not a security weakness—it is an architectural strength. It reflects the principle of bounded context and supports the operational independence that microservices are designed to achieve.

Appendix D: Microservices, Duplication, and Architectural Trade-offs

While Appendix C argues that divergence in authorisation logic across microservices is intentional and desirable, it is important to acknowledge the counterpoint: that such divergence may also signal architectural inefficiency, particularly in enterprise environments where services are developed for the same business unit.

Microservices are designed to support autonomy, scalability, and independent deployment. However, when multiple services within the same domain replicate core capabilities—such as user management, role handling, or access control—the benefits of autonomy may be outweighed by the cost of duplication. This is especially true when the services are not truly independent, but are instead modular components of a larger, tightly coupled business process.

In such cases, the question arises: does the microservice model add value, or does it simply fragment what could be a more maintainable modulithic architecture?

Duplication of core logic across services introduces:

* **Increased maintenance overhead**, as changes must be replicated and retested across multiple codebases.
* **Inconsistent behaviour**, as each service may interpret shared concepts (e.g., roles, permissions) differently.
* **Coordination burden**, when changes to shared logic require synchronised updates across teams.

This is the same architectural entanglement that the paper warns against when discussing centralised IdAM enforcement—only inverted. Instead of one system trying to do too much, multiple systems are each doing the same thing, independently.

The solution is not to centralise authorisation into a single microservice “to rule them all,” nor to push it into the IdAM. Rather, it is to recognise when microservices are being used to solve problems that are better addressed through modular design within a single deployable unit. In such cases, a modulithic architecture may offer better cohesion, clearer boundaries, and lower operational cost.

Microservices are not inherently superior. Their value depends on context. When used to support independently evolving services with distinct responsibilities, they offer clear benefits. But when applied to tightly coupled business logic within a single domain, they introduce unnecessary complexity and operational cost.

Architectural decisions must be driven by system responsibility, not by pattern popularity. The goal is not purity of model, but clarity of purpose.